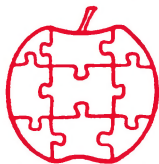


# Apple

\$1.80



# Assembly Line

---

Volume 4 -- Issue 7

April, 1984

---

## In This Issue...

|  |    |
|--|----|
| Cyclic Redundancy Check Subroutine . . . . .     | 2  |
| More Clocks for Apple. . . . .                   | 10 |
| An Evening with Woz. . . . .                     | 11 |
| Converting to Intellec Hex Format. . . . .       | 14 |
| Quick DOS Updating vs. MASTER CREATE . . . . .   | 19 |
| Burning and Erasing EPROMS . . . . .             | 23 |
| Using EXEC Files with Rak-Ware's DISASM. . . . . | 26 |
| Macro Source Code Now Available. . . . .         | 28 |

## Have we got news for you this month!

First, the simple announcements: We now have a new S-C Macro Cross Assembler for the Zilog Z-8 microprocessor. For only \$32.50 Macro Assembler owners can add the ability to develop code for this popular chip.

And some good news for you Corvus hard disk owners: The problem in the Macro Assembler with having your Target File on a different volume from your source files is now whipped. Just send in your original Version 1.1 diskette for a free update.

Now the big story: After repeated requests from many users, we have decided to make available the complete Source Code for S-C Macro Assembler Version 1.1. See the last page of this issue for details.

Another product for which we have held back selling source code is the Double Precision Floating Point package for Applesoft (DPFP). From now on that product will be sold WITH source code, at the unforgetably low price of \$50. If you already are a registered owner of DPFP, or can supply other proof-of-purchase, we will send you the source code for \$15. In case you never heard of DPFP, it is a 2048-byte &-module that provides 21-digit arithmetic and I/O for Applesoft.

## Cyclic Redundancy Check Subroutine.....Bob Sander-Cederlof

In the May 1983 AAL I wrote about checksums and parity, two ways to guarantee the integrity of data. In the world of microprocessors, you may encounter checksums at the end of data records on tape or disk, and parity bits on characters sent via a modem between computers. Tacking on parity bits and checksums to data helps in checking whether the data has been changed. However, there are more secure methods.

The best method I have ever heard of is commonly called Cyclic Redundancy Check, or CRC for short. Since it appears a lot more complicated than parity or checksum, it stands to reason it should have a more complex name. Right? But programmers have a way of reducing all complexity to three capital letters, so we will call it CRC.

First, a little review. The kind of parity I most frequently see adds an 8th bit on the left of a 7-bit value. The parity bit is chosen so that the total number of one-bits in the 8-bit byte is odd. For example, the seven bit number 1011010 (which might stand for an ASCII "Z") becomes 11011010, or \$DA. If we run into the value 01011010 (\$5A), we know there has been an error somewhere. Of course we don't know which bit is wrong, but we know at least one is because the total number of one-bits is even.

Checksums I run into are normally 8-bit or 16-bit "sums" of a large number of bytes or double bytes. I put "sums" in quotation marks because the checksum may be formed by the exclusive-or operation rather than true addition. In fact, it usually is. Eight-bit checksums formed with exclusive-or are in reality a kind of lengthwise parity. Each bit of the checksum is a parity bit for the column of bits in that position in the block of data.

In the old days, when dinosaurs first began to associate with herds of wildly spinning tape drives, you heard the words "vertical parity" and "longitudinal parity". Vertical parity was in those days a seventh bit for each six-bit character written on the tape, and longitudinal parity was a 7-bit character tacked on the end of each tape record, just like a checksum.

Enough review.

CRC is a much better scheme. A typical CRC implementation would add a 16-bit code to the end of a 256-byte block of data. A simple checksum would warn you of all single-bit errors, and some errors involving more than one bit. But CRC could detect all single and double bit errors, all errors with an odd number of error bits, all bursts of errors up to 16-bits in a row, and nearly all bursts of 17 or 18 bits in a row. Wow!

Also, you can even use CRC codes to CORRECT single-bit errors, if you trade off against some detection of longer error bursts.

S-C Macro Assembler Version 1.0.....\$80  
 S-C Macro Assembler Version 1.1 Update.....\$12.50  
 Source Code for Version 1.1 (on two disks).....\$100  
 Full Screen Editor for S-C Macro (with complete source code).....\$49  
 S-C Cross Reference Utility (without source code).....\$20  
 S-C Cross Reference Utility (with complete source code).....\$50  
 DISASM Dis-Assembler (RAK-Ware).....\$30  
 Source Code for DISASM.....additional \$30

S-C Word Processor (with complete source code).....\$50  
 Double Precision Floating Point for Applesoft (with source code).....\$50  
 S-C Documentor (complete commented source code of Applesoft ROMs).....\$50  
 Source Code of //e CX & P8 ROMs on disk.....\$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each \$15  
 Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.  
 QD#1: Oct-Dec 1980      QD#2: Jan-Mar 1981      QD#3: Apr-Jun 1981  
 QD#4: Jul-Sep 1981      QD#5: Oct-Dec 1981      QD#6: Jan-Mar 1982  
 QD#7: Apr-Jun 1982      QD#8: Jul-Sep 1982      QD#9: Oct-Dec 1982  
 QD#10: Jan-Mar 1983      QD#11: Apr-Jun 1983      QD#12: Jul-Sep 1983  
 QD#13: Oct-Dec 1983      QD#14: Jan-Mar 1984

OBJ.APWRT][F (Don Lancaster, Synergetics).....\$30  
 Quick-Trace (Anthro-Digital).....(reg. \$50) \$45  
 Visible Computer: 6502 (Software Masters).....(reg. \$50) \$45  
 ES-CAPE: Extended S-C Applesoft Program Editor.....\$60  
 Amper-Magic (Anthro-Digital).....(reg. \$75) \$65  
 Amper-Magic Volume 2 (Anthro-Digital).....(reg. \$35) \$30  
 Routine Machine (Southwestern Data Systems).....(reg. \$64.95) \$60  
 FLASH! Integer BASIC Compiler (Laumer Research).....\$79  
 Fontrix (Data Transforms).....\$75  
 Aztec C Compiler System (Manx Software).....(reg. \$199) \$180  
 IACalc Spreadsheet Program.....(reg. \$84.95) \$75  
     The one we use every day. It's better than Visicalc!  
 Locksmith 5.0 (Omega MicroWare).....(reg. \$99.95) \$90

Blank Diskettes (Verbatim).....2.50 each, or package of 20 for \$45  
     (Premium quality, single-sided, double density, with hub rings)  
 Vinyl disk pages, 6"x8.5", hold two disks each.....10 for \$6  
 Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each  
     or \$25 per 100

These are cardboard folders designed to fit into 6"x9" Envelopes.  
 Envelopes for Diskette Mailers.....6 cents each  
 ZIF Game Socket Extender (Ohm Electronics) .....\$20

Books, Books, Books.....compare our discount prices!  
 "Apple ][ Circuit Description", Gayler.....(\$22.95) \$21  
 "Understanding the Apple II", Sather.....(\$22.95) \$21  
 "Enhancing Your Apple II, vol. 1", Lancaster.....(\$17.95) \$17  
 "Incredible Secret Money Machine", Lancaster.....(\$7.95) \$7  
 "Beneath Apple DOS", Worth & Lechner.....(\$19.95) \$18  
 "Bag of Tricks", Worth & Lechner, with diskette.....(\$39.95) \$36  
 "Assembly Lines: The Book", Roger Wagner.....(\$19.95) \$18  
 "What's Where in the Apple", Second Edition.....(\$24.95) \$23  
 "What's Where Guide" (updates first edition).....(\$9.95) \$9  
 "6502 Assembly Language Programming", Leventhal.....(\$18.95) \$18  
 "6502 Subroutines", Leventhal.....(\$17.95) \$17  
 "Real Time Programming -- Neglected Topics", Foster.....(\$9.95) \$9

We have small quantities of other great books, call for titles & prices.  
 Add \$1.50 per book for US shipping. Foreign orders add postage needed.

\*\*\* S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 \*\*\*  
 \*\*\* (214) 324-2050 \*\*\*  
 \*\*\* We accept Master Card, VISA and American Express \*\*\*

You will run into CRC if you look into hard disks, or well-written modem software.

I like to write well-written programs, so I have been trying to understand CRC for some time now. A long time ago I had access to a book called "Error Correcting Codes", which is a classic. But I can't locate a copy now. I have seen numerous articles on the topic, especially in Computer Design. There was even one in Byte, Sept. 83, page 438. But I couldn't make the program in Byte work the way CRC's are supposed to, and I don't save my old Computer Design magazines.

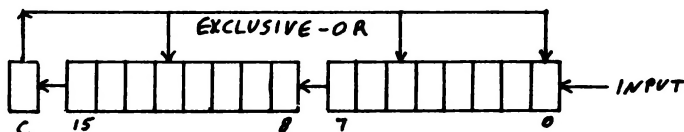
Bobby Deen to the rescue. Bobby had a copy of a public domain routine by Paul Hansknecht, of Carpenter Associates, Box 451, Bloomfield Hills, MI 48013. Actually four little subroutines, to:

- \* clear the CRC code value
- \* cycle the eight bits of a data byte through the CRC algorithm
- \* finish the CRC calculation for an outbound message
- \* check the CRC bytes of a received message.

What is the basic idea of CRC? You perform an algorithm on each bit of a block of data, and get a CRC value. You append the CRC value to the data, and transmit both data and CRC. The receiver performs the same algorithm on the total record, both the data and the CRC code; when finished, the result of the receiver's CRC algorithm should be zero. If not zero, there was an error.

I am speaking in terms of sending and receiving, as in transmitting data with a modem. It all applies equally to writing and reading records on a disk, or even in adding check codes to a ROM. The programs I wrote and will list here merely operate on a buffer in RAM, so that I can see what is happening. You can extend them to practical uses from this base.

Which brings us to algorithms. The one Bobby gave me works like this:



The 16-bit value is initialized to zero. Then each bit in the data buffer is presented one at a time where the input arrow is. The bits in the 16-bit value are all shifted left one position, and the new data bit comes in the right end to become the new bit 0. The bit which shifts out the left end is Exclusive-ORed with the bits now found in bits 12, 5, and 0. That is, if the bit shifted out was a zero, nothing happens. If the bit shifted out was a one, exclusive or the 16 bit value with \$1021.

If you understand the math of cyclic polynomials (I don't), this is supposed to be equivalent to  $X^{16} + X^{12} + X^5 + 1$ . An organization known to me only as CCITT recommends this polynomial. Another good one is reputed to be  $X^{16} + X^{15} + X^2 + 1$ , which is implemented by changing the exclusive or value from \$1021 to \$8005.

After all the bits of the data have been processed through the algorithm, 16 more zero bits are shifted through. After the zeroes, the value in the CRC register is the CRC code we append to the data.

The "receiver" processes the data the same way, starting by zeroing the CRC register. But instead of ending by shifting in 16 more zeroes, the receiver ends by shifting in the CRC code received.

I wanted to see if it really could find all those kinds of errors mentioned above. I wrote a program which would compute the CRC value and append it to a data block. Then I wrote another program which would "receive" the block and print out the resulting CRC value. Then I modified it to one-by-one toggle each bit position in the entire block, simulating a single bit error in each bit position in the whole buffer. My buffer is 256 bytes long, so that means  $8 \times 256$  or 2048, different error positions. Actually 2064, because of the two bytes of CRC.

This way I experimentally "discovered" that the value produced by the CRC computation on the received message is dependent on the error bit position. It doesn't matter what the data was. Therefore, if I had a lookup table of 2064 16-bit entries, I could search through it and find out which bit position was wrong. There must be an easier way to figure out which bit position is wrong, but that is one of the things I still need to learn.

Okay. CRC.BYTE (lines 2890-3060) is a subroutine to process the eight bits of one byte through the CRC algorithm. CRC.BYTE needs to be called once for each byte of data in the buffer, plus either two zero bytes for a SEND routine or two CRC bytes for a RECV routine.

CRC.BUFFER (lines 2700-2850) is a little subroutine which calls CRC.BYTE once for each byte in the extended buffer. I assume it is called with PNTR pointing at the first byte in the buffer, and LIMIT is equated to the byte just beyond the end. The extended buffer includes either two zeroes on the end, or the two CRC bytes.

SETUP (lines 2610-2690) is a subroutine to initialize the CRC value register to zeroes, and to set PNTR to point at the beginning of the buffer.

The SEND and RECV routines at lines 1160-1380 simulate "sending" and "receiving" the buffer. Note that both SEND and RECV finish by displaying the calculated CRC value. SEND also stores the calculated CRC value into the end of the extended

buffer. RECV should end up with a CRC value of \$0000, unless there have been bits changed between calls to SEND and RECV.

**TEST.SINGLE.BIT.ERRORS** (lines 1390-1800) is the testing subroutine which I described above. It calls **CRC.BUFFER** 2064 times. Each time a different bit is changed. I print out the resulting CRC code each time, eight to a line, with the address of the byte containing the error bit leading the line. Before running **TEST.SINGLE.BIT.ERRORS**, you should run **SEND** to be sure a valid CRC code is installed in the extended test buffer.

I wrote another test routine which tests all two-bit errors. See **TEST.DOUBLE.BIT.ERRORS**, lines 1810-2410. The only trouble is it would take about 72 hours to run, so I haven't let it go all the way. I designed it to step through every bit position in two nested loops. If both loops happen to be at the same bit position, the bit will be toggled twice resulting in no error. I designed the program to print the address of the current byte whenever there was no error.

You might experiment with error bursts of various lengths, which should take no longer than **TEST.SINGLE.BIT.ERRORS** to run.

I would really be interested in finding out how to go backwards from a non-zero received CRC value to correct single-bit errors. Is there some easy way, without either a huge table or a long computation? If any of you know how, or have an article that tells how, pass it along.

## Complete, Commented Source Code!

Our software is not only unlocked and fully copyable

...we often provide the complete source code on disk, at unbelievable prices!

**S-C Macro Assembler.** The key to unlocking all the mysteries of machine language. Combined editor/assembler with 29 commands, 20 directives. Macros, conditional assembly, global replace, edit, and more. Highest rating "The Book of Apple Software" in 1983 and 1984. \$80.

**Powerful cross-assembler modules** also available to owners of S-C Macro Assembler. You can develop software on your Apple for 6800, 6805, 6809, 68000, 8085, 8048, 8051, 1802, LSI-11, and Z-80 microprocessors. \$50 each.

**S-C Xref.** A support program which works with the S-C Macro Assembler to generate an alphabetized listing of all labels in a source file, showing with each label the line number where it is defined along with all line numbers containing references to the label. You get the complete source code for this amazingly fast program, on disk in format for S-C Macro Assembler. \$50.

**Full Screen Editor.** Integrates with the built-in line-oriented editor in the S-C Macro Assembler to provide a powerful full-screen editor for your assembly language source files. Drivers for Videx, STB80, and Apple //e 80-column boards are included, as well as standard 40-column version. Requires 64K RAM in your Apple. Complete source code on disk included. \$50.

**S-C Docu-Mentor for Applesoft.** Complete documentation of Applesoft internals. Using your ROM Applesoft, produces ready-to-assemble source code with full labels and comments. Educational, entertaining, and extremely helpful. Requires S-C Macro Assembler and two disk drives. \$50.

**S-C Word Processor.** The one we use for manuals, letters, our monthly newsletter, and whatever. 40-columns only, requires lower-case display and shiftkey mod. Works with standard DOS text files, but at super fast (100 sectors in 7 seconds). No competition to WordStar, but you get complete source code! \$50.

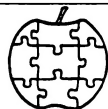
**Apple Assembly Line.** Monthly newsletter published since October, 1980, for assembly language programmers or those who would like to be. Tutorial articles, advanced techniques, handy utility programs, and commented listings of code in DOS, ProDOS, and the Apple ROMs. Helps you get the most out of your Apple! \$18/year.

**S-C SOFTWARE CORPORATION**  
2331 Gus Thomasson, Suite 125  
Dallas, TX 75228 (214) 324-2050

Professional Apple Software Since 1978

Visa, MasterCard, American Express, COD accepted.

Apple is a trademark of Apple Computer, Inc.



```

1000 *SAVE S.CRC GENERATOR (HANSKNECHT)
1010 *-----
4000- 1020 BUFFER .EQ $4000
4102- 1030 LIMIT .EQ $4102
      1040 *-----
0000- 1050 CRC .EQ $00,01
0002- 1060 PNTR .EQ $02,03
0004- 1070 TPTR .EQ $04,05
0006- 1080 TMASK .EQ $06
0007- 1090 SPTR .EQ $07,08
0009- 1100 SMASK .EQ $09
      1110 *-----
F941- 1120 PRNTAX .EQ $F941
FD8E- 1130 CROUT .EQ $FD8E
FDDA- 1140 PRBYTE .EQ $FD8E
FDED- 1150 COUT .EQ $FDED
      1160 *-----
      1170 * SIMULATE SENDING A BUFFER-FULL
      1180 *-----
0800- 20 12 09 1190 SEND JSR SETUP CLEAR CRC, POINT AT BUFFER
0803- A9 00 1200 LDA #0 CLEAR CRC BYTES IN BUFFER
0805- 8D 01 41 1210 STA LIMIT-1
0808- 8D 00 41 1220 STA LIMIT-2
080B- 20 21 09 1230 JSR CRC.BUFFER COMPUTE CRC OF 258 BYTES
080E- A6 00 1240 LDX CRC STORE CRC INTO LAST 2 BYTES
0810- A5 01 1250 LDA CRC+1
0812- 8E 01 41 1260 STX LIMIT-1
0815- 8D 00 41 1270 STA LIMIT-2
0818- 20 41 F9 1280 JSR PRNTAX DISPLAY THE CRC
081B- 4C 8E FD 1290 JMP CROUT <RETURN> AND RETURN
      1300 *-----
      1310 * SIMULATE RECEIVING A BUFFER-FULL
      1320 *-----
081E- 20 12 09 1330 RECV JSR SETUP CLEAR CRC, POINT AT BUFFER
0821- 20 21 09 1340 JSR CRC.BUFFER COMPUTE CRC OF 258 BYTES
0824- A6 00 1350 LDX CRC DISPLAY CRC IN HEX
0826- A5 01 1360 LDA CRC+1
0828- 20 41 F9 1370 JSR PRNTAX
082B- 4C 8E FD 1380 JMP CROUT
      1390 *-----
      1400 * TRY "RECEIVING" THE 258 BYTES
      1410 * WITH A KNOWN SINGLE-BIT ERROR.
      1420 *-----
      1430 TEST.SINGLE.BIT.ERRORS
082E- A9 00 1440 LDA #BUFFER
0830- 85 04 1450 STA TPTR FOR TPTR = BUFFER TO LIMIT
0832- A9 40 1460 LDA /BUFFER
0834- 85 05 1470 STA TPTR+1
0836- A5 05 1480 .1 LDA TPTR+1 PRINT TPTR "-"
0838- A6 04 1490 LDX TPTR
083A- 20 41 F9 1500 JSR PRNTAX
083D- A9 AD 1510 LDA # "-"
083F- 20 ED FD 1520 JSR COUT
0842- A9 80 1530 LDA #$80 FOR TMASK =
0844- 85 06 1540 STA TMASK $80,40,20,10,8,4,2,1
0846- A0 00 1550 .2 LDY #0
0848- B1 04 1560 LDA (TPTR),Y INVERT BIT, MAKING ERROR
084A- 45 06 1570 EOR TMASK
084C- 91 04 1580 STA (TPTR),Y
084E- 20 12 09 1590 JSR SETUP CLEAR CRC, POINT AT BUFFER
0851- 20 21 09 1600 JSR CRC.BUFFER COMPUTE CRC
0854- A9 A0 1610 LDA # " " PRINT " "CRC
0856- 20 ED FD 1620 JSR COUT
0859- A5 01 1630 LDA CRC+1
085B- A6 00 1640 LDX CRC
085D- 20 41 F9 1650 JSR PRNTAX
0860- B1 04 1660 LDA (TPTR),Y FIX ERRONEOUS BIT
0862- 45 06 1670 EOR TMASK
0864- 91 04 1680 STA (TPTR),Y
0866- 46 06 1690 LSR TMASK
0868- D0 DC 1700 BNE .2 NEXT TMASK
086A- 20 8E FD 1710 JSR CROUT ...MORE
086D- E6 04 1720 INC TPTR PRINT<CR>
086F- D0 02 1730 BNE .3 NEXT TPTR
0871- E6 05 1740 INC TPTR+1
0873- A5 04 1750 .3 LDA TPTR
0875- C9 02 1760 CMP #LIMIT
0877- A5 05 1770 LDA TPTR+1

```

```

0879- E9 41      1780      SBC /LIMIT+1
087B- 90 B9      1790      BCC .1      ...MORE
087D- 60          1800      RTS
                  1810      *-----
                  1820      TEST.DOUBLE.BIT.ERRORS
087E- A9 00      1830      LDA #BUFFER
0880- 85 07      1840      STA SPTR      FOR SPTR=BUFFER TO LIMIT
0882- A9 40      1850      LDA /BUFFER
0884- 85 08      1860      STA SPTR+1
                  1870      *-----
0886- A9 80      1880      .1      LDA #$80      FOR SMASK=80,40,20,10,8,4,2,1
0888- 85 09      1890      STA SMASK
                  1900      *-----
088A- A9 00      1910      .2      LDA #BUFFER      FOR TPTR=BUFFER TO LIMIT
088C- 85 04      1920      STA TPTR
088E- A9 40      1930      LDA /BUFFER
0890- 85 05      1940      STA TPTR+1
                  1950      *-----
0892- A9 80      1960      .3      LDA #$80      FOR TMASK=80,40,20,10,8,4,2,1
0894- 85 06      1970      STA TMASK
                  1980      *-----
0896- A0 00      1990      .4      LDY #0
0898- B1 04      2000      LDA (TPTR),Y      MAKE FIRST ERROR
089A- 45 06      2010      EOR TMASK
089C- 91 04      2020      STA (TPTR),Y
089E- B1 07      2030      LDA (SPTR),Y      MAKE SECOND ERROR
08A0- 45 09      2040      EOR SMASK
08A2- 91 07      2050      STA (SPTR),Y
08A4- 20 12      2060      JSR SETUP      CLEAR CRC, POINT AT BUFFER
08A7- 20 21      2070      JSR CRC.BUFFER      COMPUTE CRC
08AA- B1 07      2080      LDA (SPTR),Y      FIX BOTH ERRORS
08AC- 45 09      2090      EOR SMASK
08AE- 91 07      2100      STA (SPTR),Y
08B0- B1 04      2110      LDA (TPTR),Y
08B2- 45 06      2120      EOR TMASK
08B4- 91 04      2130      STA (TPTR),Y
                  2140      *-----
08B6- A5 00      2150      LDA CRC      IF CRC=0, DISPLAY POINTERS
08B8- 05 01      2160      ORA CRC+1
08BA- D0 03      2170      BNE .5      ...CRC.NE. 0, SO CONTINUE
08BC- 20 E8      2180      JSR DISPLAY.POINTERS
                  2190      *-----
08BF- 46 06      2200      .5      LSR TMASK      NEXT TMASK
08C1- D0 D3      2210      BNE .4      ...MORE
08C3- E6 04      2220      INC TPTR      NEXT TPTR
08C5- D0 02      2230      BNE .6
08C7- E6 05      2240      INC TPTR+1
08C9- A5 04      2250      .6      LDA TPTR
08CB- C9 02      2260      CMP #LIMIT
08CD- A5 05      2270      LDA TPTR+1
08CF- E9 41      2280      SBC /LIMIT+1
08D1- 90 BF      2290      BCC .3      ...MORE
                  2300      *-----
08D3- 46 09      2310      LSR SMASK      NEXT SMASK
08D5- D0 B3      2320      BNE .2      ...MORE IN THIS BYTE
08D7- E6 07      2330      INC SPTR      NEXT SPTR
08D9- D0 02      2340      BNE .7
08DB- E6 08      2350      INC SPTR+1
08DD- A5 07      2360      .7      LDA SPTR
08DF- C9 02      2370      CMP #LIMIT
08E1- A5 08      2380      LDA SPTR+1
08E3- E9 41      2390      SBC /LIMIT+1
08E5- 90 9F      2400      BCC .1      ...MORE
08E7- 60          2410      RTS
                  2420      *-----
08E8- A5 05      2430      DISPLAY.POINTERS
08EA- A6 04      2440      LDA TPTR+1      PRINT TPTR="--TMASK" ;
08EC- 20 41      2450      LDX TPTR
08EF- A9 AD      2460      JSR PRNTAX
08F1- 20 ED      2470      LDA #"--"
08F4- A5 06      2480      JSR COUT
08F6- 20 DA      2490      LDA TMASK
08F9- A9 A0      2500      JSR PRBYTE
08FB- 20 ED      2510      LDA #""
08FE- 20 ED      2520      JSR COUT
0900- A6 07      2530      LDA SPTR+1      PRINT SPTR="--SMASK"
0902- 20 41      2540      LDX SPTR
0905- A9 AD      2550      JSR PRNTAX
0907- 20 ED      2560      LDA #"--"
0907- 20 ED      2570      JSR COUT

```



----- ( ADVERTISEMENT ) -----

# ONE-KEY DOS

-----

HOW MANY TIMES HAVE YOU TYPED 'CATALOG' SO FAR THIS YEAR?  
HOW ABOUT 'DELETE'?  
OR 'LOAD' (OR 'LAOD')?

-----

I'VE USED BOB'S ASSEMBLER EVER SINCE IT BECAME AVAILABLE AND ITS THE BEST! DOS, HOWEVER, IS ANOTHER MATTER. LAST YEAR, I GOT FED UP WITH (MIS)TYPING ALL THOSE DOS COMMANDS. WAITING FOR THE RIGHT ARROW CURSOR TO TRAVERSE FILE NAMES IS A ROYAL PAIN, AS WELL. RESULT: ONE-KEY DOS. HERE'S WHAT IT'LL DO FOR YOU:

- > '-' MOVES CURSOR UP (UP-ARROW ON //E)
- > RTN MOVES CURSOR DOWN (DOWN-ARROW ON //E)
- > ':' GENERATES 'CATALOG' (= ON //E)
- > '/' GENERATES 'LIST'
- > 'O' GENERATES 'DELETE'
- > '1-9' GENERATES 'RUN', 'BRUN', 'LOAD'.. ETC
- > CTRL-I MOVES CURSOR TO END-OF-LINE (TAB ON //E)
- > 2-LINE CUSTOM TITLES FOR YOUR DISKS. TITLE LINES, FREE SPACE AND VOLUME# DISPLAYED WITH THE CATALOG (WON'T SCROLL OFF).
- > COMPLETE CATALOG SCROLL CONTROL.
- > COMPATIBLE WITH STANDARD DOS, 'ZIPPY' DOS (FAST LOADER) AND DIVERSI-DOS (FAST LOAD/SAVE).
- > BOOTUP 'HELLO' FILE NAME CAN BE CHANGED DURING CONVERSIONS.
- > MAXFILES DEFAULT VALUE (USUALLY 3) ALSO CHANGEABLE.
- > DISKS ARE RECONVERTABLE TO ANY (NON-PROTECTED) DOS WITH OR WITHOUT ONE-KEY FEATURES.
- > WHOLLY CONTAINED WITHIN NORMAL DOS MEMORY (\$9D00-BFFF).
- > COMPLETE FUNCTIONAL DISCRIPTION OF ALL DOS AREAS EFFECTED.
- > SCROLL AND EOL-TAB FEATURES WORK WITH THE S-C ASSEMBLER.  
(BOB'S CTRL-I TABBER IS EASILY CHANGEABLE; I USE CTRL-N)

-----  
UNPROTECTED CONVERTER DISK AND DECAL SET: \$17.95  
(MACHINE READABLE, PRINTABLE DOCUMENTATION ON DISK)

(IF A FRIEND HAS ALREADY GIVEN YOU THE DISK, I'LL  
SUPPLY THE DECAL SET FOR \$9 BUT YOU MUST ENCLOSE  
A SELF-ADDRESSED STAMPED ENVELOPE.  
MINIMUM SIZE 8 1/2 X 3 3/4)

-----

BROOKE W BOERING  
135 STEPHEN ROAD  
APTOS CA - 95003  
(YES, I MOVED)

----- ( ADVERTISEMENT ) -----

```

090A- A5 09 2580 LDA SMASK
090C- 20 DA FD 2590 JSR PRBYTE
090F- 4C 8E FD 2600 JMP CROUT
2610 *-----*
0912- A9 00 2620 SETUP LDA #0 CLEAR CRC
0914- 85 00 2630 STA CRC
0916- 85 01 2640 STA CRC+1
0918- A9 00 2650 LDA #BUFFER SET UP PNTR TO BUFFER
091A- 85 02 2660 STA PNTR
091C- A9 40 2670 LDA /BUFFER
091E- 85 03 2680 STA PNTR+1
0920- 60 2690 RTS
2700 *-----*
2710 * COMPUTE CRC FROM (PNTR) THRU LIMIT
2720 *-----*
2730 CRC.BUFFER
0921- A0 00 2740 .1 LDY #0 SCAN THRU THE BUFFER
0923- B1 02 2750 LDA (PNTR),Y
0925- 20 39 09 2760 JSR CRC.BYTE
0928- E6 02 2770 INC PNTR NEXT BYTE
092A- D0 02 2780 BNE .2
092C- E6 03 2790 INC PNTR+1
092E- A5 02 2800 .2 LDA PNTR CHECK LIMIT
0930- C9 02 2810 CMP #LIMIT
0932- A5 03 2820 LDA PNTR+1
0934- E9 41 2830 SBC /LIMIT
0936- 90 E9 2840 BCC .1 MORE TO GO
0938- 60 2850 RTS
2860 *-----*
2870 * COMPUTE CRC ON A SINGLE BYTE
2880 *-----*
2890 CRC.BYTE
0939- A2 08 2900 LDX #8 DO 8 BITS
093B- 0A 2910 .1 ASL MSB OF BYTE TO CARRY
093C- 26 00 2920 ROL CRC
093E- 26 01 2930 ROL CRC+1
0940- 90 0E 2940 BCC .2 --> 0, GET NEXT BIT
0942- 48 2950 PHA --> 1, TOGGLE POLYNOMIAL BITS
0943- A5 00 2960 LDA CRC
0945- 49 21 2970 EOR #$21 TOGGLE BITS 0 AND 5
0947- 85 00 2980 STA CRC
0949- A5 01 2990 LDA CRC+1
094B- 49 10 3000 EOR #$10 TOGGLE BIT 12
094D- 85 01 3010 STA CRC+1
094F- 68 3020 PLA
0950- CA 3030 .2 DEX NEXT BIT
0951- D0 E8 3040 BNE .1
0953- 60 3050 RTS
3060 *-----*

```

More Clocks for Apple.....Bob Sander-Cederlof

Some more clock cards have been brought to my attention recently.

Prometheus Versacard includes a clock, and it is compatible with ProDOS due to its ability to emulate a Thunderclock. List price is \$199.

Naturally, there is a clock on the Mountain Computer CPS/Multifunction Card. Naturally, because "CPS" stands for Clock Parallel Serial, the three functions the card supports. I cannot find a current price for this card.

Practical Peripherals is advertising ProClock, no price mentioned.

An Evening with Woz.....Bill Morgan

Well, maybe not a whole evening, but a good portion of it. And certainly not alone, there were about 150 others in the room. But I did have the opportunity to attend a dinner sponsored by the River City Apple Corps, in Austin, Texas, and hear a speech by Steve Wozniak, the designer of our favorite pastime.

Most of Steve's speech was devoted to the history of his involvement with computers, and the development of the Apple II. That story is pretty well-known by now, so I won't mention too much of it here. The most interesting facets to me were hearing how much of a prankster Woz has always been, and finding out how many features of the Apple II were motivated only by Steve's desire to write a Breakout game in BASIC.

My favorite part of the evening was the question-and-answer session and the informal chats afterward, when we all got our chance to ask about what we really wanted to know. The first question is mine, the rest came from all around the room. These are the items that seem to be of most concern to AAL readers:

How about 65816 machines?

We're heavily involved in a computer based around that chip. But, final computer becoming a full-fledged product is subject to too many other variations, such as: you start working on it and you decide, no, this computer didn't come out right, it's too long, the actual date it will be done, it's not enough, we have to do a different product. So, it may be as soon as a few months, and it may be as long as a couple years before Apple has a product based around that new processor. Fortunately it is 100% compatible with what we've done before. Meaning it's a compatible upgrade, and that's what the Apple II has to do.

When can we expect a portable //e?

It's ... in the works. We're certainly thinking about it and delving into it and unless the project gets cancelled, probably very soon, but you can never say for sure until it's out.

How about color on the Macintosh?

There is no color on the Macintosh. ... Laser printers ... (and) ... LCD displays ... are converging on black and white technology being appropriate for that product line. There is no color for the Macintosh at this time.

Do you expect to see the 3 1/2 inch disks on the //e?

Apple believes that it's time to start moving the entire company toward higher density, better technology, more modern technology disk drives, and the 3 1/2 inch disk drives from

Sony that is in the Lisa and Macintosh computers now is the proper direction to move in. It'll be interesting to see how it unfolds over time, as to how the conversion is made and yet extreme compatibility and support taken into account. All the software exists today on 5 1/4 inch disks. How do we get there?

It could be like your second disk can be a nice 3 1/2 inch with a lot more storage capability, but it may be years before it's proper to expect bootable software, to be able to boot on 3 1/2 inch drives. It's a challenge, and it just can't be turned over overnight. We could come out with a product for the Apple II today that uses a 3 1/2 inch drive as your only drive, and you know you can't run any of your software on it.... The sales of such a product would not start until there was a software base established.

What are you personally working on?

I'm interested in the future Apple II families. We're always pursuing higher performance-to-cost versions of the Apple II. And sometimes that's achieved by integrating several chips down into one custom chip, or by looking at accessories that are very commonplace, that almost everyone's going to buy for their //e. You can build one version of it with a lot of those accessories in and save a lot of money in the end, a lot of hassle. There are ways to improve the cost/performance ratio.

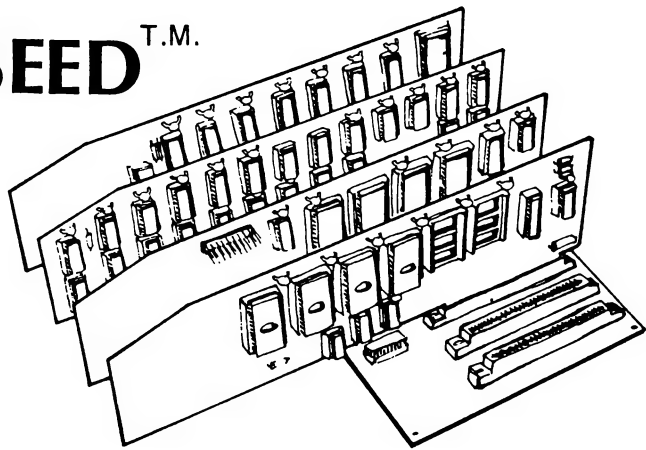
The other end, we're always trying to improve the capabilities of the machine. How are we going to eventually, someday, challenge IBM in the multi-megabyte computer world, the high-end? How are we going to improve performance?, increase screen resolution?, all those sort of questions, those sort of enhancements. I've been working very closely on one of those projects in Apple since returning.

... I think we've got to start heading towards a real, more useful home machine that does have a few of the things that we originally pursued, that we now believe is only about 10% of our market. Things such as: speech recognition and speech generation, built in, because they're relatively inexpensive and easy to do now to some level of quality. And it should also have more of the home-ish features, the features that are used in a personal, home environment built in.

So, that's the gist of it. I would like to thank Stuart Greenfield, of the River City Apple Corps, for the invitation to attend their dinner, and of course thank you, Woz, for coming to visit us.

One last note: Steve referred to a portable Apple //e as "probably very soon". Lately we've been hearing about the Apple //c, a 9-pound machine sporting 128K RAM, one disk drive, built-in serial and parallel ports, and no slots. Also no monitor, which sounds a little strange. Price -- \$1200. The //c announcement is expected in late April.

# APPLESEED<sup>T.M.</sup>



Appleseed is a complete computer system. It is designed using the bus conventions established by Apple Computer for the Apple ][. Appleseed is designed as an alternative to using a full Apple ][ computer system. The Appleseed product line includes more than a dozen items including CPU, RAM, EPROM, UART, UNIVERSAL Boards as well as a number of other compatible items. This ad will highlight the Mother board.

## BX-DE-12 MOTHER BOARD

The BX-DE-12 Mother board is designed to be fully compatible with all of the Apple conventions. Ten card slots are provided. Seven of the slots are numbered in conformance with Apple standards. The additional three slots, lettered A, B and C, are used for boards which don't require a specific slot number. The CPU, RAM and EPROM boards are often placed in the slots A, B and C.

The main emphasis of the Appleseed system is illustrated by the Mother Board. The absolute minimum amount of circuitry is placed on the Mother Board; only the four ICs which are required for card slot selection are on the mother board. This approach helps in packaging (flexibility & smaller size), cost (buy only what you need) and reparability (isolate and fix problems through board substitution).

Appleseed products are made for O.E.M.s and serious industrial/scientific users. Send for literature on the full line of Appleseed products; and, watch here, each month, for additional items in the Appleseed line.

Appleseed products are not sold through computer stores.

Order direct from our plant in California.

Apple is a registered trademark of Apple Computer, Inc.

**DOUGLAS ELECTRONICS**

**718 Marina Blvd., San Leandro, CA 94577 • (415) 483-8770**

## Converting to Intellec Hex Format.....Bob Sander-Cederlof

The Prom Burners reviewed elsewhere in this issue all were designed especially for Apple owners, and plug directly into an Apple slot. Believe it or not, there are other computers.... There are many brands of industrial grade prom burners which are not specifically designed for a particular computer host. Most of these connect to a serial port on whatever host computer you choose.

Many of these expect to receive data in a special format, called by some the Intellec Hex Paper Tape Format. Or, since at least two of those capitalized words are rather old-fashioned, the Intellec Hex Format. Intellec is also used to communicate with a variety of Emulation hardware, and Development Systems.

The S-C Assemblers produce either binary files or the binary image in memory of the object code. Can we convert a file or range of RAM to the Intellec format, and send it via a serial port? Sure, it only takes a little software....

Let's first simplify a little by assuming we can BLOAD a binary file first into Apple RAM. Then we only need a program which can translate and send a block of Apple RAM.

I would like to be able to operate the program by a control-Y monitor command. I want to type what looks like the memory move ("M") command: the first address specifies to the target system where the data should load; the second and third addresses specify the Apple RAM to be sent. I also would like to be able to specify which slot the serial port is in, or where in RAM a subroutine to send bytes to the target system can be found if there is no intelligent interface card.

The program I wrote fulfills those wishes. It loads at \$300, and self-installs a control-Y vector for the monitor. Location \$0000 and \$0001 must then be set to the low- and high-bytes of the port, and the "M"-like control-Y command can be typed. For example:

```
:BRUN B.INTELLEC
:$0:2 0
:$F800<800.FFF^O^Y
```

The port value is 0002, which means slot 2. I wrote the program so that a port value 0001 thru 0007 means a slot number; 0100 thru FFFF means a subroutine address for your own driver; 0000 means send the output where it already is directed when you type the control-Y command. The "^O^Y" on the third line above means "control-O control-Y", which is how you type a control-Y when you are in the S-C Assembler. If you type the command from the monitor (\*-prompt), omit the control-O.

I chose to send the data in a form that is compatible with both Intel and Zilog specifications, as I understand them. I do not have any equipment which expects this format around here, so I cannot test the program with live data. If you do, and you

find I have mis-interpreted something, I would sure appreciate some feedback.

There are two types of records sent: data and end-of-file records. Each record begins with a colon (:) and ends with carriage return linefeed (CRLF, which is \$8D8A). The records consist of five fields.

Record length field: two hex digits which specify how many bytes of data will be in the data field. Will be 00 for an end-of-file record. In keeping with Zilog's standard, the maximum length will be 32 bytes.

Address field: four hex digits which specify the load address in a data record, and the run address in an end-of-file record.

Record type field: 00 for a data record, and 01 for an end-of-file record.

Data field: two hex digits for each byte of data specified by the record length field. Empty for an end-of-file record.

Checksum field: two hex digits which represent the complement of the 8-bit sum of the 8-bit bytes which result from converting each pair of hex digits in the other four fields of this record to 8-bit binary values.

Lines 1250-1330 of the program set up the control-Y vector for the Apple Monitor. If this is unfamiliar to you, you might like to read all about it in the October 1981 issue of Apple Assembly Line, pages 14-17.

Briefly, once set up, a control-Y command will branch to your own code. It gives a way to extend the Apple monitor. You can type up to three addresses before the control-Y, and they will be parsed by the monitor and saved in five two-byte variables (called A1, A2, A3, A4, and A5). If you type "aaaa<bbbb.cccc" before the control-Y:

aaaa will be saved in A4 and A5  
bbbb will be saved in A1 and A3  
cccc will be saved in A2

If you wish to pass more than three items of information to the control-Y routine, you can pre-store them in other locations. In my routine, you must pre-store the port value in \$0000 and \$0001.

The whole control-Y routine is shown in just four lines of code: lines 1470-1500. Of course, these are all subroutine calls.

TURN.ON.OUTPUT.PORT (lines 1510-1650) examines locations \$0000 and 0001. If they contain 0000, then the output port is not changed. If they contain 0001 thru 00FF, the lower three bits are used to select an intelligent interface card in slot 1

through 7. A larger value indicates your own driver routine address.

TURN.OFF.OUTPUT.PORT (lines 2010-2030) sets the output back to the Apple screen.

SEND.DATA.RECORDS (lines 1660-1890) divides the area to be transmitted into a number of 32-byte blocks. Each block is send as one data record. The final block may be less than 32 bytes.

SEND.EOF.RECORD (lines 1900-2000) sends the end-of-file record. The original loading address is assumed to be the run address. If you would rather send 0000 for a run address, you can change lines 1960 and 1980 to "LDA #0".

SEND.RECORD (lines 2050-2330) formats and transmits one record of either type, using the count, address, and type information already setup by the caller. It also updates A1 and A4 for the next record.

SEND.BYTE (lines 2340-2420) accumulates a byte in the checksum, and then converts it to two hex digits and transmits it.

You can use this program with any of the S-C Macro Assemblers or Cross Assemblers, exactly as shown. If you are using some other brand of assembler, you will probably have to leave the assembler environment to load this program, load the object code you wish to transmit, and run the program.

```

1000 *SAVE S.INTELLEC HEX FORMATTER
1010 .OR $300
1020 -----
0000- 1030 PORT .EQ $00,01
0002- 1040 CHECK.SUM .EQ $02
0003- 1050 TYPE .EQ $03
0004- 1060 COUNT .EQ $04
0005- 1070 REMAINING .EQ $05,06
1080 -----
003C- 1090 A1 .EQ $3C,3D
003E- 1100 A2 .EQ $3E,3F
0040- 1110 A3 .EQ $40,41
0042- 1120 A4 .EQ $42,43
0044- 1130 A5 .EQ $44,45
1140 -----
03F8- 1150 CTRL.VECTOR .EQ $3F8 THRU $3FA
03EA- 1160 DOS.REHOOK .EQ $3EA
1170 -----
FCB4- 1180 MON.NXTA4 .EQ $FCB4
FD8E- 1190 MON.CROUT .EQ $FD8E
FDDA- 1200 MON.PRHEX .EQ $FDDA
FDED- 1210 MON.COUT .EQ $FDED
FE93- 1220 MON.SETVID .EQ $FE93
1230 -----
1240 * SETUP CONTROL-Y
1250 -----
0300- A9 10 1260 SETUP LDA #SEND.DATA
0302- 8D F9 03 1270 STA CTRL.VECTOR+1
0305- A9 03 1280 LDA /SEND.DATA
0307- 8D FA 03 1290 STA CTRL.VECTOR+2
030A- A9 4C 1300 LDA #$4C
030C- 8D F8 03 1310 STA CTRL.VECTOR
030F- 60 1320 RTS

```



```

1330 *-----
1340 *  #0:XX YY  (LO,HI OF PORT)
1350 *  *TARGET<START.END<Y>
1360 *      IF PORT IS 0, DO NOT CHANGE OUTPUT
1370 *      IF PORT IS 1...7, OUTPUT TO SLOT.
1380 *      ELSE OUTPUT TO SUBROUTINE
1390 *      SEND BYTES START...END
1400 *
1410 *      1.  TURN ON OUTPUT PORT
1420 *      2.  SEND DATA RECORDS
1430 *      3.  SEND EOF RECORD
1440 *      4.  TURN OFF OUTPUT PORT
1450 *-----
1460 SEND.DATA
0310- 20 1C 03 1470 JSR TURN.ON.OUTPUT.PORT
0313- 20 35 03 1480 JSR SEND.DATA.RECORDS
0316- 20 61 03 1490 JSR SEND.EOF.RECORD
0319- 4C 73 03 1500 JMP TURN.OFF.OUTPUT.PORT
1510 *-----
1520 TURN.ON.OUTPUT.PORT
031C- A6 01 1530 LDX PORT+1      HI-BYTE OF PORT SPECIFIED
031E- D0 0A 1540 BNE .1
0320- A5 00 1550 LDA PORT      LO-BYTE, MUST BE SLOT
0322- 29 07 1560 AND #$07
0324- F0 0E 1570 BEQ .3      SLOT 0, JUST SCREEN
0326- 09 C0 1580 ORA #$C0
0328- D0 03 1590 BNE .2
032A- 8A 1600 .1 TXA      ...ALWAYS
032B- A6 00 1610 LDY PORT      HI-BYTE OF SUBROUTINE
032D- 85 37 1620 .2 STA $37      LO-BYTE OF SUBROUTINE
032F- 86 36 1630 STX $36
0331- 20 EA 03 1640 JSR DOS.REHOOK
0334- 60 1650 .3 RTS
1660 *-----
1670 SEND.DATA.RECORDS
0335- A9 00 1680 LDA #0
0337- 85 03 1690 STA TYPE
0339- E6 3E 1700 INC A2      POINT JUST BEYOND THE END
033B- D0 02 1710 BNE .1
033D- E6 3F 1720 INC A2+1
033F- 38 1730 .1 SEC
0340- A2 20 1740 LDX #32
0342- A5 3E 1750 LDA A2      SEE HOW MANY BYTES LEFT
0344- E5 3C 1760 SBC A1
0346- 85 05 1770 STA REMAINING
0348- A5 3F 1780 LDA A2+1
034A- E5 3D 1790 SBC A1+1
034C- 85 06 1800 STA REMAINING+1
034E- D0 08 1810 BNE .2      USE MIN(32,A2-A1+1)
0350- E4 05 1820 CPX REMAINING
0352- 90 04 1830 BCC .2
0354- A6 05 1840 LDX REMAINING
0356- F0 08 1850 BEQ .3      ...FINISHED
0358- 86 04 1860 .2 STX COUNT
035A- 20 79 03 1870 JSR SEND.RECORD
035D- 4C 3F 03 1880 JMP .1      ...ALWAYS
0360- 60 1890 .3 RTS
1900 *-----
1910 SEND.EOF.RECORD
0361- A0 00 1920 LDY #0
0363- 84 04 1930 STY COUNT
0365- C8 1940 INY
0366- 84 03 1950 STY TYPE
0368- A5 44 1960 LDA A5      RUN ADDRESS (LO)
036A- 85 42 1970 STA A4
036C- A5 45 1980 LDA A5+1      RUN ADDRESS (HI)
036E- 85 43 1990 STA A4+1
0370- 4C 79 03 2000 JMP SEND.RECORD
2010 *-----
2020 TURN.OFF.OUTPUT.PORT
0373- 20 93 FE 2030 JSR MON.SETVID
0376- 4C EA 03 2040 JMP DOS.REHOOK
2050 *-----

```

```

0379- A9 BA 2060 SEND.RECORD
037B- 20 ED FD 2070 LDA #*:
037E- A9 00 2080 JSR MON.COUT
0380- 85 02 2090 LDA #0
0382- A5 04 2100 STA CHECK.SUM
0384- 20 B7 03 2110 LDA COUNT
0387- A5 43 2120 JSR SEND.BYTE
0389- 20 B7 03 2130 LDA A4+1
038C- A5 42 2140 JSR SEND.BYTE
038E- 20 B7 03 2150 LDA A4
0391- A5 03 2160 JSR SEND.BYTE
0393- 20 B7 03 2170 LDA TYPE
0396- A5 04 2180 JSR SEND.BYTE
0398- F0 0F 2190 LDA COUNT
039A- A0 00 2200 BEQ .2
039C- B1 3C 2210 LDY #0
039E- 20 B7 03 2220 LDA (A1),Y
03A1- C8 2230 JSR SEND.BYTE
03A2- 20 B4 FC 2240 INY
03A5- C6 04 2250 JSR MON.NXTA4
03A7- D0 F3 2260 DEC COUNT
03A9- 38 2270 BNE .1
03AA- E5 02 2280 SEC
03AC- 20 B7 03 2290 SBC CHECK.SUM
03AF- 20 8E FD 2300 JSR SEND.BYTE
03B2- A9 8A 2310 JSR MON.CROUT
03B4- 4C ED FD 2320 LDA #$8A LINEFEED
03B7- 48 2330 JMP MON.COUT
03B8- 18 2340 *-----
03B9- 65 02 2350 SEND.BYTE
03BB- 85 02 2360 PHA
03BD- 68 2370 CLC
03BE- 4C DA FD 2380 ADC CHECK.SUM
03C0- 2410 STA CHECK.SUM
03C2- 2400 PLA
03C4- 2410 JMP MON.PRHEX
03C6- 2420 *-----

```

### OBJ.APWRT] [F

Detailed, complete, and thorough disassembly script includes full details on customizing Applewriter IIe and capturing your own source code for modification.

Two unlocked, jammed-full and double-sided diskettes plus a free "must have" bonus book for only \$29.95.

*Now six sides*

SOURCECODE  
% Synergetics  
Box 1300-AAL  
Thatcher, AZ 85552  
(602) 428-4073

VISA and MASTERCARGE accepted.

## Quick DOS Updating vs MASTER.CREATE.....Bob Sander-Cederlof

When DOS was young, Apples tended to have varying amounts of memory under 48K. Some had 16K, which was the standard purchase at a computer store; others 24K, with one row of 16K and two of 4K; others 32K; and some 48K. Trying to write a DOS image that would fit all of these memories was quite a task.

Apple introduced the concept of a "master" and a "slave" disk. Master disks have a generic image of DOS. The boot process first loads the DOS image as though the machine only has 16K RAM, and then the image is relocated as high as possible in memory. Slave disks have a frozen image, already relocated for a particular memory size. The INIT command always creates a slave disk. In order to make a master disk you either copy and old master using COPYA (or equivalent copy program), or you use the MASTER.CREATE program on the DOS System Master Disk. (For a while the MASTER.CREATE program was called UPDATE 3.3.)

But now! But now you will have a difficult time finding an Apple with less than 48K memory. After all, the chips are only about a dollar apiece, or \$8 to \$12 for a set of eight. Who needs master disks anymore?

A lot of people think they do, because MASTER.CREATE is there and the reference manual makes such a big deal about it. And this causes a problem. What if I want a master disk with a modified DOS? MASTER.CREATE always reads the DOS image off the system master disk, and it is unmodified. Well, you can use a disk zap program on a copy of the system master.

Or, you can forget all about MASTER.CREATE and use my handy-dandy little patch installer. The program which follows reads the DOS image from the first 3 tracks into memory from \$4000 thru \$64FF. Then it installs patches from a table of patches; this part is almost identical to the patch installer published in the April 1983 issue of AAL. Finally it writes the patched DOS back on the first three tracks. And it does all this so fast you'll think it never happened.

Once you have coded the patches you want, and have tested them, you can update all your old DOS 3.3 disks almost as fast as you can open and close the drive door. With slight modifications, you could have it write the patched image on successive disks without re-reading and re-patching each time.

Looking at the program, Lines 1200-1240 do the overall job. Just below that, lines 1260-1290 give two entry points to a block of code that sets up an IOB for RWTS and then calls RWTS. The only difference between the two calls is the opcode, either READ or WRITE. Below that point, there is a backwards loop that counts from track 2, sector 4, back to track 0, sector 0. Just for fun, I print out the track and sector numbers just before reading or writing each sector. (If you get tired of the fun, simply delete line 1450, the JSR \$F941.)

The DOS image on tracks 0, 1, and 2 is not in exactly the same order as you find it in memory after booting. Therefore the

patcher maps patch addresses to the new locations. Lines 1060-1080 define the remapping constants. Addresses which in the running image will be between \$B600 and \$BFFF will be located from \$4000 thru \$49FF. If the original was a master, code which does the relocating part of the boot will be found from \$4A00 thru \$4BFF. The code between \$9D00 and \$B5FF will be found from \$4C00 thru \$64FF. The two constants DOS.9D and DOS.B6 are used in figuring the application points of the patches in lines 2110, 2350, and 2540.

For a full explanation of lines 1590-1900, see the April 1983 AAL, pages 24-27. The patch set up to be installed in lines 2020-2580 is the fast LOAD, BLOAD, RUN, BRUN patch from pages 2-8 of the same issue.

```

                                1000 *SAVE S.APPLY DOS PATCHES
                                1010 *-----
0000-                          1020 PNTR      .EQ $00,01
0002-                          1030 PATCH     .EQ $02,03
0004-                          1040 SECTOR.CNT .EQ $04
                                1050 *-----
4000-                          1060 DOS.IMAGE .EQ $4000 - $64FF
5100-                          1070 DOS.9D   .EQ $9D00-DOS.IMAGE-$0C00
7600-                          1080 DOS.B6   .EQ $B600-DOS.IMAGE
                                1090 *-----
03E3-                          1100 GETIOB   .EQ $3E3
03D9-                          1110 RWTS     .EQ $3D9
                                1120 *-----
B7E8-                          1130 IOB      .EQ $B7E8
B7EB-                          1140 IOB.VOLUME .EQ IOB+3
B7EC-                          1150 IOB.TRACK .EQ IOB+4
B7ED-                          1160 IOB.SECTOR .EQ IOB+5
B7F0-                          1170 IOB.BUFADR .EQ IOB+8
B7F4-                          1180 IOB.OPCODE .EQ IOB+12
                                1190 *-----
                                1200 PATCH.DOS
0800- 20 0A 08 1210          JSR READ.DOS.IMAGE
0803- 20 52 08 1220          JSR PATCHER
0806- 20 0D 08 1230          JSR WRITE.DOS.IMAGE
0809- 60          1240          RTS
                                1250 *-----
                                1260 READ.DOS.IMAGE
080A- A9 01 1270          LDA #$01          READ OPCODE
080C- 2C          1280          .HS 2C
                                1290 WRITE.DOS.IMAGE
080D- A9 02 1300          LDA #$02          WRITE OPCODE
080F- 8D F4 B7 1310          STA IOB.OPCODE
0812- A9 00 1320          LDA #0
0814- 8D F0 B7 1330          STA IOB.BUFADR
0817- 8D EB B7 1340          STA IOB.VOLUME
081A- A9 64 1350          LDA #DOS.IMAGE/256+16+16+5-1
081C- 8D F1 B7 1360          STA IOB.BUFADR+1
081F- A9 02 1370          LDA #2          TRACK 2
0821- 8D EC B7 1380          STA IOB.TRACK
0824- A9 04 1390          LDA #4          SECTOR 4
0826- 8D ED B7 1400          STA IOB.SECTOR
0829- A9 25 1410          LDA #16+16+5
082B- 85 04 1420          STA SECTOR.CNT
082D- AD EC B7 1430          LDA IOB.TRACK .1
0830- AE ED B7 1440          LDY IOB.SECTOR
0833- 20 41 F9 1450          JSR $F941
0836- 20 E3 03 1460          JSR GETIOB
0839- 20 D9 03 1470          JSR RWTS
083C- AC ED B7 1480          LDY IOB.SECTOR
083F- 88          1490          DEY
0840- 10 05 1500          BPL .2
0842- A0 0F 1510          LDY #15
0844- CE EC B7 1520          DEC IOB.TRACK
0847- 8C ED B7 1530          STY IOB.SECTOR .2
084A- CE F1 B7 1540          DEC IOB.BUFADR+1
084D- C6 04 1550          DEC SECTOR.CNT
084F- D0 DC 1560          BNE .1
0851- 60          1570          RTS

```

----- APPLE SOFTWARE -----

**NEW!!! FONT DOWNLOADER & EDITOR (\$39.00)**

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Can be used with every word processor capable of sending ESC and control codes to the printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed, emphasized, underlined, etc.) apply to custom fonts. Full HIRES screen editor lets you create your own custom characters and special graphics symbols. Compatible with many 'dumb' & 'smart' printer I/F cards. User driver option provided. Specify printer: Apple Dot Matrix Printer, C.Itoh 8510A (Prowriter), Epson FX-80/100 or OkiData 92/93.

**DISASM 2.2e - AN INTELLIGENT DISASSEMBLER (\$30.00)**

Investigate the inner workings of machine language programs. DISASM converts 6502 machine code into meaningful, symbolic source. Creates a standard DOS 3.3 text file which is directly compatible with DOS ToolKit, LISA and S-C (4.0 and MACRO) assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor & Pg Zero pg names included.) An address-based cross reference table provides further insight into the inner workings of machine language programs. DISASM is an invaluable machine language learning aid to both the novice & expert alike. **SOURCE code: \$60.00**

**S-C ASSEMBLER (Ver4.0 only) SUPPORT UTILITY PACKAGE (\$30.00)**

\* SC.XREF - Generates a GLOBAL LABEL Cross Reference Table for complete documentation of source listings. Formatting control accommodates all printer widths for best hardcopy outputs. \* SC.GSR - Global Search and Replace eliminates tedious manual renaming of labels. Search all or part of source. Optional prompting for user verification. \* SC.TAB - Tabulates source files into neat, readable form. **SOURCE code: \$40.00**

----- HARDWARE/FIRMWARE -----

**THE 'PERFORMER' CARD (\$39.00)**

Plugs into any Apple slot to convert your 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu provides easy access to printer fonts. Eliminates need to remember complicated ESC codes and key them in to setup printer. Added features include perforation skip, auto page numbering with date & title. Also includes large HIRES graphics screen dump in normal or inverse plus full page TEXT screen dump. Specify printer: Epson MX-80 with Graftrax-80, MX-100, MX-80/100 with GraftraxPlus, NEC 80923A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. Oki bonus: print EMPHASIZED & DOUBLE STRIKE fonts! **SOURCE code: \$30.00**

**FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM (\$25.00)**

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Three basic modes: Dumb Terminal, Remote Console & Programmable Modem. Added features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back option and built-in printer buffer. Supports most 80-column displays and the 1-wire shift key mod. Uses a superset of Apple's Comm card and Micromodem II commands. A-C hardware differences prevent 100% compatibility with Comm card. **SOURCE code: \$60.00**

**RAM/ROM DEVELOPMENT BOARD (\$30.00)**

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip. Use a 6116 type RAM chip for program development or just extra memory. Plug in a preprogrammed 2716 EPROM to keep your favorite routines 'on-line'. A versatile board with many uses! Maps into \$C000-CnFF and \$C800-CFFF memory space. Circuit diagram included.

**NEW!!! SINGLE BOARD COMPUTER KIT (\$20.00)**

Kit includes etched PC board (with solder mask and plated thru holes) and assembly instructions. User provides 6502 CPU, 6116 2K RAM, 6821 dual 8-bit I/O and 2732 4K EPROM plus misc common parts. Originally designed as intelligent printer interface - easily adapted to many applications needing dedicated controller. (Assembled and tested: \$119.00)

All assembly language SOURCE code is fully commented & provided in both S-C Assembler & standard TEXT formats on an Apple DOS 3.3 diskette. Specify your system configuration with order. Avoid a \$3.00 postage and handling charge by enclosing full payment with order (MasterCard & VISA excluded). Ask about our products for the VIC-20 and Commodore 64!

**R A K - W A R E**

**41 Ralph Road      West Orange      NJ      07052      (201) 325-1885**

```

1580 *-----
1590 PATCHER
0852- A9 85 1600 LDA #PATCHES-1
0854- 85 00 1610 STA PNTR
0856- A9 08 1620 LDA /PATCHES-1
0858- 85 01 1630 STA PNTR+1
085A- A0 00 1640 LDY #0
1650
085C- 20 7D 08 1660 .1 JSR GET.BYTE LENGTH OF NEXT PATCH
085F- F0 1B 1670 BEQ .4 FINISHED
0861- AA 1680 TAX SAVE LENGTH IN X
0862- 20 7D 08 1690 JSR GET.BYTE ADDRESS OF PATCH
0865- 85 02 1700 STA PATCH
0867- 20 7D 08 1710 JSR GET.BYTE
086A- 85 03 1720 STA PATCH+1
1730
086C- 20 7D 08 1740 .2 JSR GET.BYTE
086F- 91 02 1750 STA (PATCH),Y
0871- E6 02 1760 INC PATCH
0873- D0 02 1770 BNE .3
0875- E6 03 1780 INC PATCH+1
0877- CA 1790 .3 DEX
0878- D0 F2 1800 BNE .2
087A- F0 E0 1810 BEQ .1 ... ALWAYS
1820
087C- 60 1830 .4 RTS
1840 *-----
1850 GET.BYTE
087D- E6 00 1860 INC PNTR
087F- D0 02 1870 BNE .1
0881- E6 01 1880 INC PNTR+1
0883- B1 00 1890 .1 LDA (PNTR),Y
0885- 60 1900 RTS
1910 *-----
1920 PATCHES
1930 *-----
1940 * S.FAST LOAD
1950 *
1960 * FAST "LOAD" AND "BLOAD"
1970 *
1980 *
1990 * INSTALLED IN UNUSED AREAS IN DOS 3.3:
2000 * $BA69-$BA95 (45 BYTES FREE)
2010 * $BCDF-$BCFF (33 BYTES FREE)
2020 *-----
AC96- 2020 READ.RANGE .EQ $AC96
B0B6- 2030 READ.NEXT.SECTOR .EQ $B0B6
B36F- 2040 END.OF.DATA.ERROR .EQ $B36F
B5C1- 2050 RANGE.LENGTH .EQ $B5C1,C2
B5C3- 2060 RANGE.ADDRESS .EQ $B5C3,C4
B5CB- 2070 BUFFER.ADDRESS .EQ $B5CB,CC
B5E4- 2080 SECTOR.COUNT .EQ $B5E4,E5
B5E6- 2090 BYTE.OFFSET .EQ $B5E6
2100 *-----
0886- 2C 69 44 2110 .DA #P1.LENGTH,$BA69-DOS.B6
2120 .PH $BA69
BA69- AD E6 B5 2130 PATCH1 LDA BYTE.OFFSET LAST BYTE OF
BA6C- D0 24 2140 BNE GO.READ.RANGE A SECTOR?
BA6E- AD C2 B5 2150 LDA RANGE.LENGTH+1 WHOLE SECTOR LEFT?
BA71- F0 1F 2160 BEQ GO.READ.RANGE NO.
BA73- AD CB B5 2170 LDA BUFFER.ADDRESS SAVE BUFFER ADDRESS
BA76- 48 2180 PHA
BA77- AD CC B5 2190 LDA BUFFER.ADDRESS+1
BA7A- 48 2200 PHA
BA7B- AD C3 B5 2210 LDA RANGE.ADDRESS READ DIRECTLY
BA7E- 8D CB B5 2220 STA BUFFER.ADDRESS INTO RANGE
BA81- AD C4 B5 2230 LDA RANGE.ADDRESS+1
BA84- 8D CC B5 2240 STA BUFFER.ADDRESS+1
2250 READ.LOOP
BA87- 20 B6 B0 2260 JSR READ.NEXT.SECTOR
BA8A- B0 03 2270 BCS .1
BA8C- 4C DF BC 2280 JMP PATCH2
BA8F- 4C 6F B3 2290 .1 JMP END.OF.DATA.ERROR
2300 GO.READ.RANGE
BA92- 4C 96 AC 2310 JMP READ.RANGE
002C- 2320 P1.LENGTH.EQ #-PATCH1
2330 .EP
2340 *-----

```

```

08B5- 21 DF 46 2350      .DA #P2.LENGTH,$BCDF-DOS.B6
                                2360      .PH $BCDF
BCDF- EE E4 B5 2370  PATCH2 INC SECTOR.COUNT
BCE2- D0 03 2380      BNE .1
BCE4- EE E5 B5 2390      INC SECTOR.COUNT+1
BCE7- EE C4 B5 2400 .1    INC RANGE.ADDRESS+1      NEXT PAGE
BCEA- EE CC B5 2410      INC BUFFER.ADDRESS+1
BCED- CE C2 B5 2420      DEC RANGE.LENGTH+1
BCF0- D0 0B 2430      BNE .2
BCF2- 68 2440      PLA      RESTORE BUFFER
BCF3- 8D CC B5 2450      STA BUFFER.ADDRESS+1
BCF6- 68 2460      PLA
BCF7- 8D CB B5 2470      STA BUFFER.ADDRESS
BCFA- 4C 96 AC 2480      JMP READ.RANGE
                                2490
BCFD- 4C 87 BA 2500      .2      JMP READ.LOOP
0021-      2510  P2.LENGTH.EQ *-PATCH2
                                2520      .EP
                                2530  *-----
08D9- 03 A5 5B 2540      .DA #P3.LENGTH,$ACA5-DOS.9D
                                2550      .PH $ACA5
ACA5- 4C 69 BA 2560  PATCH3 JMP PATCH1
0003-      2570  P3.LENGTH.EQ *-PATCH3
                                2580      .EP
                                2590  *-----
08DF- 00      2600      .DA #0      END OF PATCHES

```

Burning and Erasing EPROMs.....Bob Sander-Cederlof

We get a lot of questions about EPROM burners and erasers.  
Perhaps this list will help...

### Burners

PROM Blaster System, \$119, Apparat, 4401 South Tamarac Parkway, Denver, CO 80237. Phone (303) 741-1778 or (800) 525-7674. Will burn most 24-pin EPROMS. Price includes personality modules for 2704, 2708, 2508, 2758, 2716(TI), 2516, 2716, 2532, 2732, 2732A, 68764, 2815, and 2816. ZIF socket for EPROM. No power switch, so you must power down the Apple whenever you insert or remove an EPROM.

Apple-PROM, \$149.95, Computer Technology Associates, 1704 Moon N.E., Suite 14, Albuquerque, NM 87112. Phone (505)298-0942. Will burn most 24-pin EPROMS. DIP switch selection for 2708, 2716, 2516, 2532, 2732, 2732A, 2764, 2564. Low insertion force socket for EPROM.

Romwriter, \$175, Mountain Computer....(I cannot find any recent ads, but they are still listed in distributor catalogs). We have heard that they are no longer manufacturing this card, but there are still many available. Only burns 2716 (single voltage version, not TI). ZIF for EPROM. Power switch on card allows you to safely insert and remove EPROMs without turning off your Apple. I have been using this one for several years with no problems, although I did rewrite the software to suit my own tastes and needs.

Quick EPROM Writer, \$149, available from Handwell Corp., 4962 El Camino Real, Suite 119, Los Altos, CA 94022. Phone (415) 962-9265. Made in Taiwan by "COPAM". Burns both 24- and 28-pin EPROMs. All software is in firmware on the card. Nice menu select for 2716, 2516, 2532, 2732, 2732A, 2564, 2764, and 27128. No personality modules or switch selection required, as all configuration is software controlled. Power is applied to and removed from the ZIF socket under software control, so that EPROMs can be inserted and removed without turning off your Apple. Manual includes schematic, pinout diagrams for EPROMs, and a (sparsely) commented listing of firmware. The firmware apparently implements an intelligent burning algorithm, which burns twice as long as it takes to get the byte burned, rather than using a fixed delay for each byte. The result is much faster burn times than most other burners listed here.

HM3264, \$395, Hollister Microsystems, 5081 Fairview, Hollister, CA 95023. Phone (408) 637-0753. Programs 2716, 2732, 2732A, 2764, and 27128. Henry Spragens uses this one, and says it is very well designed and built, though expensive. Henry has modified the software Hollister provides to use the intelligent burn algorithm (it was pretty slow until he did this). Hollister use the C800-CFFF address space, like Mountain Computer does, as a 2048-byte window into the EPROM. Bank switching on the card lets you program larger EPROMs. Power switch on card allows you to safely insert and remove chips. A program switch helps prevent inadvertent programming.

Model EP-2A-79, \$169 plus \$17 to \$35 each for personality modules and \$19 to \$40 for software. Optimal Technology, Earlysville, VA 22936. Phone (804) 973-5482. Programs full range from 2708 through 27128, plus 38E70 and 8751 MPUs, assuming you purchase the corresponding personality modules and software. It is not clear to me whether this plugs directly into an Apple or requires a separate serial interface card.

### Erasers

QUV-T8 EPROM Erasers, Logical Devices, 1321E N.W. 65 Place, Fort Lauderdale, FL 33309. Phone (305) 974-0967 or (800) EE1-PROM (that is 331-7766). Four models, ranging from \$49.95 to \$124.95. I use and recommend the \$97.50 model, which includes a slide-out tray, anti-static foam pad, UV indicator lens, timer, and safety interlock switch.

Spectronics, marketed by JDR Microdevices, 1224 S. Bascom Avenue, San Jose, CA 95128. Phone (800) 662-6279 or (408) 995-5430. Six models from \$83 to \$595. The \$83 unit has no timer, all the others do. [ JDR's latest ad in Byte shows eight 250nsec 4116's for \$7.95! ]

Jade Computer Products carries both brands of EPROM Erasers. Their price on the least expensive Spectronics is only \$69.95.

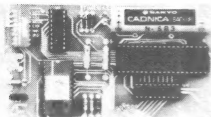
Jameco Electronics lists an eraser for \$79.95.



## That's Why We're So Good At It!



**Automatically date stamps files with PRO-DOS**



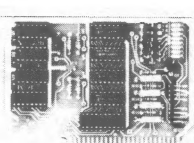
**NEW 1984  
DESIGN**  
An official  
**PRO-DOS Clock**

- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- NiCad battery keeps the TIMEMASTER II running for over ten years.
- Full emulation of ALL other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the TIMEMASTER to emulate other clocks, we just drop off features. That's why we can emulate others, but others CAN'T emulate us.
- The TIMEMASTER II will automatically emulate the correct clock card for the software you're using. You can also give the TIMEMASTER II a simple command to tell it which clock to emulate (but you'll like the TIMEMASTER mode better). This is great for writing programs for those who don't understand the thought some other clock card.
- Basic Machine Code, CP/M, and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time (many examples are included).
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The **TIMEMASTER II** includes 2 disks with some really fantastic time oriented programs (over 40) including appointment book so you'll never forget to do anything again. Enter your appointments up to a year in advance then forget them. Plus DOS data so it will automatically add the date when disk files are created or modified. The disk is over a \$200.00 value alone—we give the software others sell. All software packages for business, data base management and communications are made to read the **TIMEMASTER II**. If you want the most powerful and the easiest to use clock for your Apple, you want a **TIMEMASTER II**.

**PRICE \$129.00**

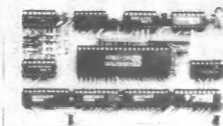
## Super Music Synthesizer Improved Hardware and Software



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
  - It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
  - Now with new improved software for the easiest and the fastest music input system available anywhere.
  - We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
  - Easy to program in Basic, to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
  - Four white noise generators which are great for sound effects.
  - Plays music in true stereo as well as two discrete quadraphonic.
  - Full control of attack, volume, decay, sustain and release.
  - Will play songs written for ALF synthesizer (ALF software will not take advantage of all of our card's features. Their software sounds the same in our synthesizer).
  - Our card will play notes from 30Hz to beyond human hearing.
  - Automatic shutoff on power-up or if reset is pushed.
  - Many many more features.
- PRICE \$159.00**

**PRICE \$159.00**

## Z-80 PLUS!



- **TOTALLY compatible with ALL CP/M software.**
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- **RUNS WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PLOT and ALL other CP/M software with no preboot.**
- A semi-custom I.C. and a low parts count allow the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at **DOES 4MHZ.**)
- **DOES EVERYTHING** the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

**PRICE \$139.00**

## Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- **TOTALLY Videx Compatible.**
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

|                   | PRICE       | BUILT IN<br>SOFTWARE | SHIRT-LESS<br>SUPPORT | VIDEO<br>DEMO | 800 CALLING<br>NUMBER | 7-DIGIT<br>HOTLINE | SHIRTLESS<br>SUPPORT | 800<br>OVERLINE | UNIVERSAL<br>CARRIER |
|-------------------|-------------|----------------------|-----------------------|---------------|-----------------------|--------------------|----------------------|-----------------|----------------------|
| <b>VIEWMASTER</b> | <b>179</b>  | <b>YES</b>           | <b>YES</b>            | <b>YES</b>    | <b>YES</b>            | <b>YES</b>         | <b>YES</b>           | <b>YES</b>      | <b>YES</b>           |
| <b>SUPERTRAK</b>  | <b>MORE</b> | <b>NO</b>            | <b>YES</b>            | <b>NO</b>     | <b>NO</b>             | <b>NO</b>          | <b>YES</b>           | <b>YES</b>      | <b>YES</b>           |
| <b>WARDWARD</b>   | <b>MORE</b> | <b>NO</b>            | <b>NO</b>             | <b>NO</b>     | <b>NO</b>             | <b>YES</b>         | <b>NO</b>            | <b>NO</b>       | <b>NO</b>            |
| <b>VISION-80</b>  | <b>MORE</b> | <b>YES</b>           | <b>YES</b>            | <b>NO</b>     | <b>NO</b>             | <b>YES</b>         | <b>NO</b>            | <b>NO</b>       | <b>NO</b>            |
| <b>OWN-VISION</b> | <b>MORE</b> | <b>NO</b>            | <b>YES</b>            | <b>NO</b>     | <b>NO</b>             | <b>NO</b>          | <b>NO</b>            | <b>NO</b>       | <b>YES</b>           |
| <b>VIEW-MARU</b>  | <b>MORE</b> | <b>YES</b>           | <b>YES</b>            | <b>NO</b>     | <b>NO</b>             | <b>YES</b>         | <b>NO</b>            | <b>NO</b>       | <b>YES</b>           |
| <b>SMARTER</b>    | <b>MORE</b> | <b>YES</b>           | <b>YES</b>            | <b>NO</b>     | <b>NO</b>             | <b>NO</b>          | <b>YES</b>           | <b>YES</b>      | <b>NO</b>            |
| <b>VIDEOTRY</b>   | <b>MORE</b> | <b>NO</b>            | <b>NO</b>             | <b>YES</b>    | <b>NO</b>             | <b>YES</b>         | <b>YES</b>           | <b>NO</b>       | <b>YES</b>           |

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

**PRICE \$179.00**

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95 K storage in 80 columns! The 64K config. is all that's needed. 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.

### MemoryMaster IIe 128K RAM Card

- Precision software disk emulation for Basic, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

**MemoryMaster IIe with 128K**

**Upgradeable MemoryMaster IIe with 64K**  
**Non-Upgradeable MemoryMaster IIe with**

### Non-Upgradeable MemoryMaster IIe with 64K

Our boards are far superior to most of the other computer electronics made today. JILLIC's are high quality sockets with mid-spacer components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE II, III, II+ and Franklin. The MemoryMaster IIc is the only Apple IIc engineering also manufactures a full line of data acquisition and control products for the Apple A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

**Send Check or Money Order to:**

**APPLIED ENGINEERING**

**P.O. Box 798**

**Carrollton, TX 75006**

**Call (214) 492-2027**

8 a.m. to 11 p.m. 7 days a week  
MasterCard, Visa & C.O.D. Welcome  
No extra charge for credit cards

**Texas Residents Add 5% Sales Tax**  
**Add \$10.00 If Outside U.S.A.**  
**Dealer Inquiries Welcome**

[ Bob is the author of DISASM, owner of Rak-Ware ]

I recently received a phone call from Alan Lloyd who had just purchased DISASM. He wanted to use it to disassemble the Autostart ROM so he could customize the code for a particular application. He was frustrated by the limited editing capabilities of DISASM which makes you start all over again if you don't catch your mistake before hitting RETURN. Since he had to enter the starting and ending addresses of over a dozen data tables, he began searching for an easier (and less painful) way of entering the data. He decided to try using an EXEC file with DISASM, and it worked! Well, sort of.

I thought about the problems he ran into, and found out some interesting things about the S-C Macro Assembler along the way. It turns out that with the help of a small patch to DISASM that it is possible to run the entire program via "remote control" using an EXEC file.

The first step is to create the TEXT file that will later be EXECed. You can do this in a word processor, if your word processor makes ordinary DOS text files. Or you can write an Applesoft program to help you build an array of addresses and the proper answers to the various prompts in DISASM, and then write a complete EXEC file. I decided to use the S-C Macro Assembler, because you can use the TEXT <filename> command to write a text file. You can have the assembler in the language card, DISASM at \$800, the thing to be disassembled wherever you want, and pop back and forth fast as lightning.

Just enter each line of "source" as if you were responding to the questions put to you by DISASM. You can even include lines to turn on display of DOS commands and I/O (MONIOC), and the BLOADing of DISASM and NAMETABLE.

The S-C Macro Assembler does make one thing difficult. Some of the questions asked by DISASM require a null line (a RETURN all by itself), and S-C makes it very hard to get a null line. The first of these is used to terminate the entry of data table addresses. (Alan was satisfied to have his EXEC file stop here and take over manually.)

Normally, S-C does not let you enter totally empty lines. Typing a line number without any following text is one of the ways to DELETE a line, just as in both BASIC's. After a little experimenting I discovered a trick to fool the S-C input routine. I still don't get a totally empty line, but I can put extra RETURNS into an existing line. Here's how:

1. Type in the text of all the non-null lines you want in your EXEC file.
2. Use the EDIT command to insert extra RETURNS in the proper places: move the cursor to the character position desired, and type ctrl-O followed by RETURN to insert each

extra RETURN. Each extra RETURN will show up as an inverse "M" as you are editing. Then type one more RETURN to exit the EDIT mode.

The next problem I ran into was the Y/N responses for the "Full Cross-Reference" and "Generate Text File" questions. DISASM looks directly at the keyboard for those two responses, so it is blind to any EXEC file inputs. A five byte patch fixes all that, so you can use EXEC file as well as keyboard inputs. Just change the code starting at location \$C5A from AD 00 C0 10 FB to 20 18 FD 09 80.

The following arbitrary example illustrates how an EXEC file might look when typed into the S-C assembler (extra RETURNS are indicated by <M>):

```
1000 MONIOC
1010 BLOAD DISASM
1020 BLOAD NAMETABLE
1030 $800G          (Use call 2048 to EXEC from BASIC)
1040 2              (select S-C Assembler format)
1050 F800           (starting physical address)
1060 F9B9           (ending physical address)
1070 F800           (starting execution address)
1080 F8CD           (table #1 start)
1090 F8CF           (table #1 end)
1100 3              (table #1 format)
1110 F962           (table #2 start)
1120 F9A5           (table #2 end)
1130 5              (table #2 format)
1140 F9A6           (table #3)
1150 F9B3
1160 8
1170 F9B4           (table #4)
1180 F9B9
1190 6
1200 <M>2000        (end of tables, and NAMETABLE address)
1210 0              (no printer output)
1220 <M>NYDEMO      (RETURN for no single cross reference,
                    N for no full cross reference,
                    Y for creating a textfile named DEMO)
```

(Of course, you realize that the explanatory comments in parentheses are not supposed to be typed.) I advise you to SAVE the lines on a file as S-C source code, using the SAVE <filename> command. This will become the copy you re-LOAD when you want to make changes. Then use the TEXT <filename> command to write out the EXEC file. Finally, EXEC <filename> to run the disassembly!

When EXECing, the table addresses are entered at a blinding speed that is almost impossible to follow. If your text file has an error in it such that it does not conform to the DISASM input syntax, then things can go very wrong very fast. For those of you who would rather not see things move along quite so fast, I suggest adding a small patch to the COUT vector which provides a short delay. The following program works fine:

```

$300:48      PHA
          A9 80      LDA #$80
          20 A8 FC    JSR $FCA8
          68          PLA
          4C F0 FD    JMP $FDF0

```

You can hook this into DOS from the assembler by typing "\$36:00 03 N 3EAG". Then change line 1030 above to \$812G (or CALL 2048+18 for EXEC from BASIC) to bypass DISASM's effort to setup the default DOS vectors.

Or you can even include all this stuff along with the original EXEC file. Either way, it is easier to use DISASM with an EXEC file when there are lots of data tables to be entered and you have fumble-fingers at the keyboard.

From now on, DISASM will be shipped with the five-byte patch indicated above already installed, and with two sample EXEC files designed to be EXECed from BASIC.

**Macro Source Code Now Available.....Bob Sander-Cederlof**

We have finally become convinced that we should make the source code of our S-C Macro Assembler available for purchase. Many of you have requested, for a long time now. We have resisted, I suppose through a mild case of the same paranoia which causes so many other software publishers to use copy protection and license agreements (which we eschew).

We have absolutely no experiential basis for mistrust. You have all treated our previous offerings of source code in the most honorable fashion, and we expect you will continue to do so.

Effective immediately, registered owners of Version 1.1 of the S-C Macro Assembler can purchase the source code for \$100. You will be able to assemble it to obtain a paper listing, study it to learn techniques, and modify it to your own tastes. We hope many of you will make improvements and send them back to us for inclusion in future versions.

The code resides on two nearly-full diskettes. You need at least two drives to assemble it. The source is fully commented, and is organized in a logical easy-to-follow manner.

If you do not yet own Version 1.1, you may purchase or upgrade to it simultaneously with the purchase of the source code, if you wish. If you are one of those who purchased the Version 4.0 source code, we will give you \$40 credit toward the purchase of the Macro 1.1 source.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$12 postage for other countries. Back issues are available for \$1.50 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)